Express Mail No: EV336586047US

UNITED STATES PATENT APPLICATION

FOR

SYSTEM AND METHOD FOR FIRMWARE TO EXPORT PRE-BOOT DATA INTO THE OPERATING SYSTEM RUNTIME ENVIRONMENT

Inventors:

Michael A. Rothman
Vincent J. Zimmer

Prepared by:

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP
12400 Wilshire Boulevard
Los Angeles, CA 90025-1026
(408) 720-8300

SYSTEM AND METHOD FOR FIRMWARE TO EXPORT PRE-BOOT DATA INTO THE OPERATING SYSTEM RUNTIME ENVIRONMENT

FIELD OF THE INVENTION

[0001] An embodiment of the present invention relates generally to computer systems and, more specifically, to bridging information from the pre-boot environment to the operating system run time environment.

BACKGROUND INFORMATION

[0002] Various events may take place during boot of a computer system. Often the user does not wish to view these events, especially if there are no failures. Sometimes, however, it would be desirable to view the boot log.

[0003] During boot, the system evolves from power on to discovery of memory, configuration and then launching of devices. The boot phase is complete once the "Boot Target," typically an operating system (OS) has launched. In state of the art systems, the OS has no visibility into the boot log. Thus, once the system boot is complete, the boot log is unavailable.

[0004] Many things can trigger an event to be logged during initialization. For instance, suppose during initialization of a hardware device, or something else related to the motherboard, an error occurs. An error event is logged. Also, a warning or a temperature setting can trigger an event to be logged. These events are logged in a pre-boot event log area.

[0005] Users may wish to view the log to determine if there are any problems with the system. There is no current standards-based mechanism to view this log information today. Today, if errors occur, an administrator must go into the pre-boot set up environment and view the log. In a more complex server environment, there might be a remote agent that parses through the error log while the system is in pre-boot.

BRIEF DESCRIPTION OF THE DRAWINGS

[0006] The features and advantages of the present invention will become apparent from the following detailed description of the present invention in which:

[0007] Figures 1A and 1B are flow diagrams of an exemplary method for exporting pre-boot data and displaying data in a standardized browser;

[0008] Figure 2 is a block diagram of an exemplary motherboard having a processor requiring boot up, according to prior art;

[0009] Figure 3 is a block diagram of an exemplary server having an out-of-band monitor, according to the prior art;

[0010] Figure 4 is a block diagram of an exemplary computing system allowing export of pre-boot data; and

[0011] Figure 5 is a block diagram showing an exemplary extensible firmware interface (EFI) configuration table in memory.

DETAILED DESCRIPTION

[0012] An embodiment of the present invention is a system and method relating to the export of pre-boot data to the runtime environment. In at least one embodiment, the pre-boot data is intended to be displayed using a standards-based browser. Enabling the export of pre-boot data to be accessible by a runtime system allows the pre-boot phase of a system to be more carefully monitored. The present system and method provides a standard mechanism to export all of the pre-boot environment data and make it available to the OS runtime environment. Effectively, the information is collected and instead of, or in addition to, storing it into some proprietary location in the firmware, embodiments of the present invention also create a memory buffer that is labeled as a runtime memory buffer so it is reserved by the firmware and not overwritten by the OS. This memory location may be described in a configuration table.

[0013] Reference in the specification to "one embodiment" or "an embodiment" of the present invention means that a particular feature, structure or characteristic described in connection with the embodiment is included in at least one embodiment of the present invention. Thus, the appearances of the phrase "in one embodiment" appearing in various places throughout the specification are not necessarily all referring to the same embodiment.

[0014] Referring to the drawings, and more particular to Figures 1A and 1B, there is shown a flow diagram of an exemplary method 600 for exporting pre-boot data and displaying data in a standardized browser. Specifically, Figure 1A is directed toward the pre-boot phase and Figure 1B is directed toward the runtime phase. When a system is reset, the interface which receives progress data, status data, error logging information,

and general system information is registered with the error logging routine, in block 602. Early system initialization continues, i.e., memory discovery, bus enumeration, etc., in block 604.

[0015] When a logging event occurs, as determined by decision block 606, the event is logged into a memory-based repository, in block 610. It is determined whether initialization is complete, in decision block 608. If not, pre-boot continues at block 604. When initialization is complete, the variety of data collected during event logging is copied to a portion of memory accessible by the operating system (OS), in block 612. In one embodiment, the logging repository of data is registered with the EFI Configuration Table 620. The configuration table serves as a bucket of globally unique identifier (GUID)/pointer pairs, pointing to the data. A runtime buffer may be established to which the configuration table will have a pointer. The OS is then launched. The OS uses the pointers in the configuration table to retrieve the data.

[0016] Referring to Figure 1B, the event data in the Configuration Table 620 is retrieved by an OS agent, in block 622. In one embodiment, the event data is converted into a series of XML pages. In some embodiments, a series of alerts or other actions can be performed, based on the data retrieved from the pre-boot event log, in block 624. For instance, if a temperature reading falls within a specific range, an alert may be sent to an administrator. This range might be outside of the failure range, so systems without the ability to export pre-boot events would boot successfully and no one would be aware that the system was in danger of overheating.

[0017] It is determined whether the event log is to be displayed in decision block 626. If not, system operation continues with runtime, as usual, in block 630. If so, then the event data is displayed in block 628, for perusal by the user. In one embodiment, the OS agent built a series of XML pages. In this embodiment, the XML pages are loaded and displayed 640 with a browser. XML pages have an advantage that most computing systems today have built-in browsers capable of viewing XML. It will be apparent to one skilled in the art that Hypertext Markup Language (HTML), Standard Generalized Markup Language (SGML), or similar formatting languages could be used to format the display. Further, for systems without a built-in browser, a simple textual or ASCII display could be used.

[0018] Referring now to Figure 2, there is shown a block diagram of an exemplary motherboard of the prior art, having a processor requiring boot up. A motherboard 100 has at least one processor 102 connected to a non-volatile memory for storing a boot sequence 106, via a front side bus 108. The boot sequence is frequently stored in flash random access memory (RAM) and in PC architecture is called the basic input/output system (BIOS). The BIOS is connected to a vital product data (VPD) storage location 104 via an I2C backplane 110. Access to the VPD is often limited by its proprietary nature. Event logging during pre-boot stores event data into the VPD. The VPD is not accessible by the operating system. Utilities that run during pre-boot are required to access and view the pre-boot event data.

[0019] The BIOS saves the error log into an area of the motherboard called the VPD. The event logger stores specific configuration information or error logging information. The VPD is typically constructed of proprietary static RAM (SRAM) that would be used

by the motherboard and its associated firmware. The VPD is not an OS discoverable resource. It is typically accessed via GPIO (general protection IO) operations, which are proprietary. Some implementations have alternate storage locations where the information is stored on the flash itself.

[0020] Referring now to Figure 3, there is shown a block diagram of an exemplary server 200 having an out-of-band monitor. Some servers are monitored via an out-of-band connection by a remote application 210. In this example, a server has at least one processor 204 connected to flash RAM 206, a baseboard management controller (BMC) 208 and a VPD or other flash memory 202. The BMC communicates with the remote application 210 via an out-of-band connection, and has access to the event log stored in the VPD/flash 202 via the BMC. In order for the remote application to retrieve the event log, it must be programmed to access the proprietary data store 202. Further, the access occurs during pre-boot. There is no standard method for retrieving this information by the server's OS during runtime.

[0021] Two classes of firmware are prevalent today: legacy firmware and IPF (Itanium Family of Processors) firmware running the extensible firmware interface (EFI) firmware interface. The present system and method establishes a standard mechanism of communication between pre-boot and the operating system for event data. A standard set of application programming interfaces (API's) including standard handshake and handoff processes between pre-boot and OS runtime are used. Certain constructs are established, so there are means to extend the firmware functionality in order to take advantage of other properties of the firmware.

[0022] Referring now to Figure 4, there is shown a block diagram of an exemplary computing system allowing export of pre-boot data. Processor 312 communicates with a memory controller hub (MCH) 302, also known as Northbridge, via the front side bus 304. The MCH 302 communicates with system memory 310 via a memory bus 306. The system memory has a reserved location for a configuration table 360, typically for an EFI Configuration Table. It will be apparent to one skilled in the art that storage in available memory within an ACPI (Advanced Configuration Power Interface) table may be used as well. The MCH 302 may also communicate with an advanced graphics port (AGP) 308 via a graphics bus 310. The MCH 302 communicates with an I/O controller hub (ICH) 320, also known as Southbridge, via a peripheral component interconnect (PCI) bus 322. A baseboard management controller 350 may be connected to the processor via a low pin count (LPC) bus. The firmware hub 352 having both BIOS 354 and VPD 356 is typically connected to the processor via the LPC, as well. The processor may be operatively connected to a network 330 via a network port 340 through the ICH 320.

[0023] Referring now to Figure 5, there is shown a block diagram illustrating an exemplary extensible firmware interface (EFI) table in memory. In one embodiment, a globally unique identifier (GUID) pointer 402 points to a location in memory 310 which holds the EFI configuration table 360. The GUID may be a 128-bit unique identifier. Both the pre-boot environment and the OS can access the table identified by the GUID. During pre-boot, events are logged into the proprietary VPD, or other non-volatile memory.

[0024] Firmware allocates some memory resource to be reserved for the event log. As the events occur, they are stored in the VPD and shadowed in a memory buffer. At some point in time prior to launching the OS, the BIOS determines that it has completed logging information. The BIOS then registers the buffer. Events may occur right up until booting the operating system.

[0025] An event logging handler is registered by the BIOS. This registration allows a protocol, or means by which event handlers or other processes in the system can call a common procedure for a common purpose. The registered handler is to be called by other processes, modules or drivers. The event logging handler is registered with the firmware that notifies all other event handlers to call this registered handler when they exhibit events. Registering an event handler installs it as a common catch-all for events that might occur. If a logging event occurs, then the event handler catches the event and logs data into the memory-based repository, or buffer, via the instructions in the event logging handler. Logging continues until pre-boot is complete.

[0026] The event data is then registered into the EFI configuration table. This table is similar to the ACPI table export. Until initialization is complete, the size required for this memory buffer is unknown. In one embodiment, the buffer is registered when it is full, and no more events are to be logged in it, just prior to launching the OS. Prior to launching the OS, the events are copied to a registered location in the EFI configuration table 360. When the OS boots, it uses the GUID pointer 402 to locate the event log in RAM. Once the events are available to the OS, they can be displayed, reported, saved, or transmitted, as desired by the user.

[0027] The eXtensible Markup Language (XML) can be used to map private firmware metadata for OS usage. The OS, in turn, can use a sophisticated presentation management engine, such as the Internet Explorer web browser, to view the data. This allows for simple, concise firmware encoding of binary data to be ballooned into verbose, ASCII XML for the OS, on the fly. The system and method described herein saves flash space in pre-boot, but allows for interoperability with future standards.

[0028] The techniques described herein are not limited to any particular hardware or software configuration; they may find applicability in any computing, consumer electronics, or processing environment. The techniques may be implemented in hardware, software, or a combination of the two. The techniques may be implemented in programs executing on programmable machines such as mobile or stationary computers, personal digital assistants, set top boxes, cellular telephones and pagers, consumer electronics devices (including DVD players, personal video recorders, personal video players, satellite receivers, stereo receivers, cable TV receivers), and other electronic devices, that may include a processor, a storage medium readable by the processor (including volatile and non-volatile memory and/or storage elements), at least one input device, and one or more output devices. Program code is applied to the data entered using the input device to perform the functions described and to generate output information. The output information may be applied to one or more output devices. One of ordinary skill in the art may appreciate that the invention can be practiced with various system configurations, including multiprocessor systems, minicomputers, mainframe computers, independent consumer electronics devices, and the like. The invention can also be practiced in distributed computing environments where tasks may be performed by remote processing devices that are linked through a communications network.

[0029] Each program may be implemented in a high level procedural or object oriented programming language to communicate with a processing system. However, programs may be implemented in assembly or machine language, if desired. In any case, the language may be compiled or interpreted.

[0030] Program instructions may be used to cause a general-purpose or special-purpose processing system that is programmed with the instructions to perform the operations described herein. Alternatively, the operations may be performed by specific hardware components that contain hardwired logic for performing the operations, or by any combination of programmed computer components and custom hardware components. The methods described herein may be provided as a computer program product that may include a machine readable medium having stored thereon instructions that may be used to program a processing system or other electronic device to perform the methods. The term "machine readable medium" used herein shall include any medium that is capable of storing or encoding a sequence of instructions for execution by the machine and that cause the machine to perform any one of the methods described herein. The term "machine readable medium" shall accordingly include, but not be limited to, solid-state memories, optical and magnetic disks, and a carrier wave that encodes a data signal. Furthermore, it is common in the art to speak of software, in one form or another (e.g., program, procedure, process, application, module, logic, and so on) as taking an action or causing a result. Such expressions are merely a shorthand way of stating the execution of the software by a processing system cause the processor to perform an action of produce a result.

[0031] While this invention has been described with reference to illustrative embodiments, this description is not intended to be construed in a limiting sense. Various modifications of the illustrative embodiments, as well as other embodiments of the invention, which are apparent to persons skilled in the art to which the invention pertains are deemed to lie within the spirit and scope of the invention.